

DOSSIER PROJET

Marine MIGNOT

2026 - Centre Européen de Formation



FORUM DE DISCUSSION ET D'INFORMATION  
SUR LE MONDE DE MICKEY

# SOMMAIRE

## 1. Introduction

## 2. Presentation du projet

2.1 Contexte	5
2.2 Objectif du projet	6
2.3 Besoins auxquels le projet répond	7
2.4 Spécificités techniques	8
2.5 Spécificités fonctionnelles	8

## 3. Analyse et conception

3.1 Analyse du besoin	9
3.2 User (utilisateurs types)	9
3.3 Cahier des charges fonctionnel	10
3.4 Conception technique	11
3.5 Maquettes et ergonomie	12

## 4. Développement

4.1 Mise en place de l'environnement	14
4.2 Développement du backend	14
4.3 Développement du frontend	17
4.4 Développement de l'administration	17
4.5 Sécurité	18

## 5. Tests et validation

5.1 Objectif des tests	20
5.2 Méthodologie utilisée	20
5.3 Validation de la sécurité	22
5.4 Recette utilisateur	22
5.5 Corrections et améliorations	23

## **6. Documentation**

6.1 Objectif de la documentation	24
6.2 Documentation technique	24
6.3 Documentation utilisateur	28
6.4 Documentation administrateur	31

## **7. Bilan personnel**

7.1 Compétences acquises	38
7.2 Difficultés rencontrées	39
7.3 Améliorations possibles	39

## **8. Conclusion**

## **9. Annexes**



## 1. INTRODUCTION

Ce dossier s'inscrit dans le cadre du passage d'un Titre Professionnel et présente le travail réalisé durant la conception et le développement d'un site web communautaire.

L'objectif principal de ce projet est de mettre en pratique l'ensemble des connaissances et compétences acquises au cours de ma formation, à travers la réalisation d'une application complète et fonctionnelle.

Le développement de ce site m'a permis d'appliquer les notions essentielles du développement web telles que l'architecture logicielle, la gestion des utilisateurs, la sécurité, ainsi que la conception d'interfaces adaptées aux besoins d'un public cible.

## 2. PRÉSENTATION DU PROJET

### 2.1 Contexte

Dans le cadre de ma formation visant l'obtention d'un Titre Professionnel, il m'a été demandé de réaliser un projet complet afin de mettre en pratique les compétences acquises en développement web.

- Installer et configurer son environnement de travail en fonction du projet web ou web mobile
- Maquetter des interfaces utilisateur web ou web mobile
- Réaliser des interfaces utilisateur statiques web ou web mobile
- Développer la partie dynamique des interfaces utilisateur web ou web mobile
- Mettre en place une base de données relationnelle
- Développer des composants d'accès aux données SQL et NoSQL
- Développer des composants métier côté serveur
- Documenter le déploiement d'une application dynamique web ou web mobile

J'ai choisi de concevoir un site web dédié aux fans du monde de Mickey, un univers particulièrement riche en histoires, personnages et intrigues. Depuis plusieurs générations, ce monde a inspiré une multitude de récits, d'interprétations et de créations qui alimentent encore aujourd'hui la curiosité des passionnés. Pourtant, malgré cet intérêt constant, la communauté reste dispersée sur de nombreuses plateformes : réseaux sociaux, groupes privés ou forums généralistes. Cette dispersion rend difficile le partage organisé des connaissances, des théories ou des analyses approfondies.

Le projet *Fan de Lore – Univers de Mickey* répond à ce besoin en proposant un espace centralisé et structuré, entièrement pensé pour favoriser les échanges. Le forum permet aux utilisateurs de discuter de leurs personnages préférés, d'explorer des éléments narratifs méconnus, de partager leurs propres découvertes ou encore de débattre autour des nombreuses histoires existantes. L'interface, simple et intuitive, facilite la navigation entre les différentes catégories et encourage une participation active.

L'objectif est de créer un véritable point de rencontre pour tous ceux qui souhaitent approfondir le lore ou simplement échanger avec d'autres passionnés. Plus qu'un lieu de discussion, le site ambitionne de devenir une base communautaire solide où chacun peut contribuer, apprendre et transmettre sa vision de cet univers emblématique.

---

## 2.2 Objectifs du projet

Le projet répond à plusieurs objectifs :

### **Objectifs pédagogiques**

- Mettre en œuvre les compétences acquises durant la formation (backend, frontend, architecture, base de données, sécurité...).
- Concevoir un projet complet depuis l'analyse du besoin jusqu'aux tests.
- Appliquer les bonnes pratiques de développement, de documentation et de gestion de projet.

## Objectifs fonctionnels

- Proposer une plateforme permettant aux utilisateurs de publier et commenter sur des sujets.
  - Offrir un environnement sécurisé pour la gestion des comptes et des contenus.
  - Permettre à un administrateur de gérer les utilisateurs, les sujets et modérer les publications.
- 

### 2.3 Besoins auxquels le projet répond

Le projet vise à répondre à plusieurs besoins identifiés :

- Un espace d'expression dédié : permettre aux fans de partager et débattre sur un sujet.
  - Une organisation claire des contenus : les catégories et discussions doivent être facilement consultables et structurées.
  - Une communauté modérée : assurer un cadre respectueux et sécurisé grâce à une interface d'administration.
  - Une gestion fiable des utilisateurs : inscription, connexion sécurisée et possibilité de modifier son mot de passe.
-

## 2.4 Spécificités techniques

Le projet repose sur les technologies suivantes :

- Langage : PHP 8.1+
- Framework : Symfony 6
- Base de données : MySQL 8
- Environnement de développement : Symfony CLI
- Tests : PHPUnit
- Service mail : Mailtrap

Ces choix techniques permettent de disposer d'un environnement robuste, sécurisé et conforme aux standards actuels du développement web.

---

## 2.5 Spécificités fonctionnelles

Le site web comprend les fonctionnalités suivantes :

### **Côté utilisateur**

- Création de compte et authentification sécurisée.
- Consultation des sujets et commentaires.
- Ajout de sujets.
- Ajout de commentaires.
- Modification de commentaires.

### **Côté administrateur**

- Gestion des utilisateurs (création, modification et suppression).
- Modération des sujets et commentaires.
- Accès à une interface dédiée.

## 3. ANALYSE ET CONCEPTION

### 3.1 Analyse du besoin

Avant de démarrer le développement, une analyse détaillée du besoin a été réalisée afin de comprendre les attentes du projet et de définir clairement les fonctionnalités essentielles.

Le site devait permettre :

- d'assurer une interaction entre utilisateurs via les commentaires,
- de permettre une gestion complète des comptes et profils,
- de garantir un cadre sécurisé et administré,
- d'être évolutif et facilement maintenable.

Cette analyse a conduit à la définition d'un ensemble de fonctionnalités prioritaires et secondaires, organisées en modules logiques pour structurer le développement.

---

### 3.2 User (utilisateurs types)

Pour comprendre les besoins des utilisateurs finaux, plusieurs profils types ont été identifiés :

- Utilisateur standard : souhaite consulter et publier des commentaires
- Administrateur : supervise l'ensemble du site, modère les contenus et gère les utilisateurs

### *Exemple de création d'utilisateur.*

```
1 $user = new User();
2         $user->setUsername($username)
3             ->setEmail($email)
4             ->setRoles(['ROLE_USER'])
5             ->setPassword($this->passwordHasher->hashPassword
($user, $password))
6                 ->setIsActive(true)
7                 ->setIsVerified(true);
```

---

### 3.3 Cahier des charges fonctionnel

Le cahier des charges fonctionnel a été structuré autour des modules suivants :

#### **Module utilisateur**

- Inscription / Connexion / Déconnexion
- Réinitialisation du mot de passe
- Espace personnel (informations)

#### **Module théories**

- Création, modification et suppression d'un sujet
- Consultation des commentaires publiés
- Système de commentaires attachés à chaque sujet

## Module administrateur

- Tableau de bord
- Gestion des utilisateurs
- Modération des catégories, sujets et commentaires

*Exemple d'accès à une route par rôle.*

```
1  {% if is_granted('ROLE_ADMIN') %} 
2      <li class="admin">
3          <a href="{{ path('admin_dashboard') }}"
4              class="{{ app.request.attributes.get('_route') == 'admin_dashboard' ?
5                  'active' : '' }}>
6                  Tableau administrateur
7              </a>
8          </li>
9      {% endif %}
```

---

### 3.4 Conception technique

#### Architecture logicielle

Le projet suit une architecture MVC (Model – View – Controller) propre à Symfony.

Elle assure une séparation claire entre :

- les données (entités & repository),
- la logique métier (services, contrôleurs),
- et la présentation (templates Twig).

Les relations entre les entités ont été définies de manière cohérente (OneToMany, ManyToOne).

*Exemple de relation entre User et Comment.*

```
1  #[ORM\OneToMany(mappedBy: 'authorUser', targetEntity:  
Comment::class, cascade: ['persist', 'remove'])]  
2      private Collection $comments;
```

---

### 3.5 Maquettes et ergonomie

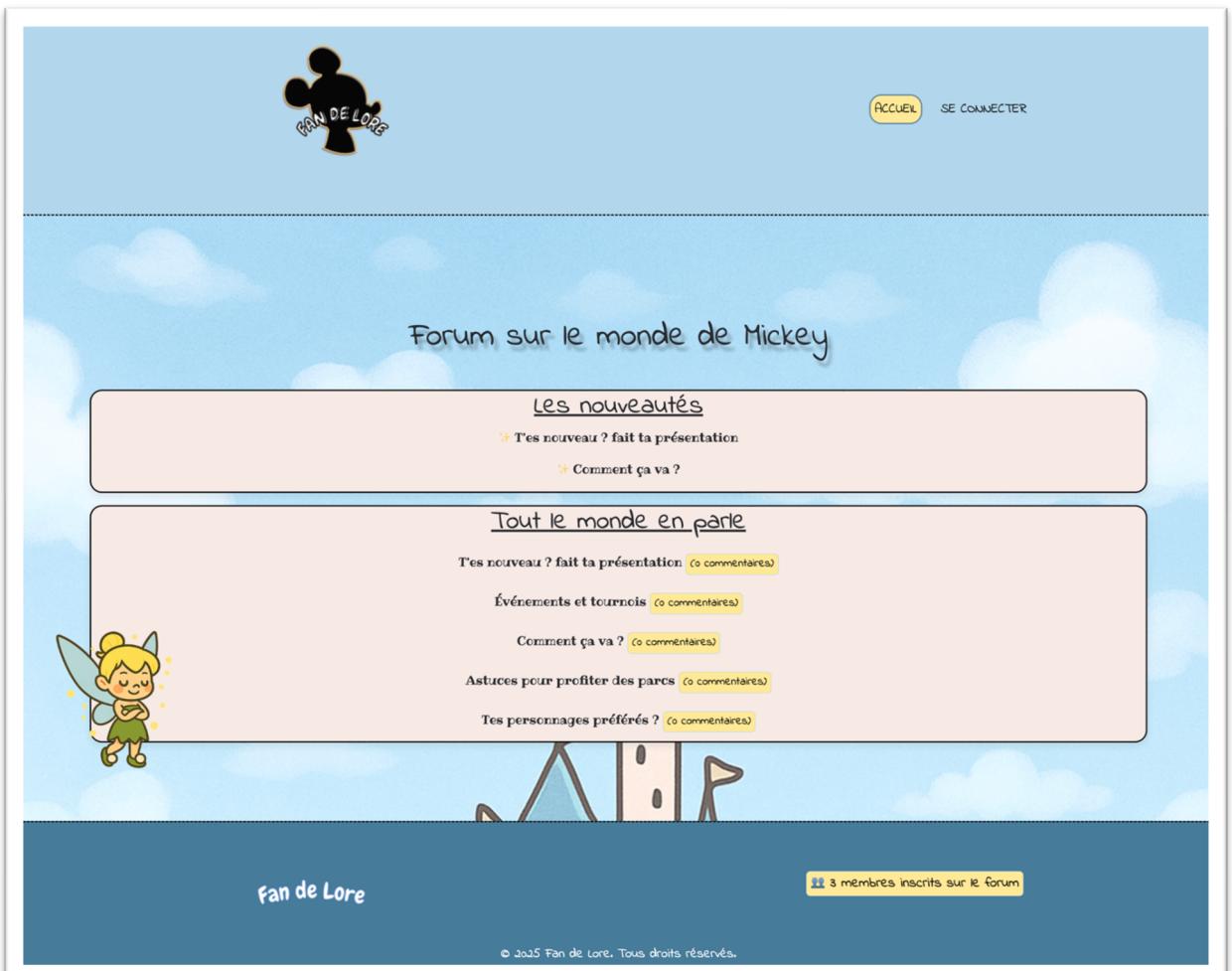
Des maquettes simples ont été réalisées afin de :

- définir la structure des pages principales,
- organiser les contenus de manière lisible,
- assurer une expérience utilisateur intuitive.

Les maquettes couvrent notamment :

- la page d'accueil,
- la page forum avec les catégories et leurs sujets,
- la page de lecture d'un sujet,
- l'espace utilisateur,
- le panneau d'administration.

*Exemple de la maquette de la page d'accueil non connectée pour la version Desktop (faite sur Figma).*



## 4. DÉVELOPPEMENT

### 4.1 Mise en place de l'environnement

Le développement a été réalisé dans un environnement entièrement configuré pour respecter les bonnes pratiques du framework Symfony.

La stack utilisée comprend :

- Symfony 6 pour la structure applicative
- PHP 8.1+
- MySQL 8 pour la base de données
- Composer pour la gestion des dépendances
- Symfony CLI pour le serveur local
- Git pour la gestion de version
- Mailtrap pour le test des emails
- PHPUnit pour les tests unitaires

Cette configuration a permis un cycle de développement fluide, sécurisé et conforme aux standards modernes.

---

### 4.2 Développement du backend

#### Gestion des utilisateurs

La première étape a été la mise en place du système d'authentification :

- Inscription sécurisée (hashage des mots de passe via PasswordHasher)
- Connexion avec gestion des sessions
- Rôles utilisateurs (ROLE\_USER, ROLE\_ADMIN)
- Réinitialisation du mot de passe avec un jeton sécurisé

*Exemple pour montrer l'utilisation de PasswordHasher.*

```
1  if ($form->isSubmitted() && $form->isValid()) {  
2      $plainPassword = $form->get('plainPassword')  
3      ->getData();  
4      $user->setPassword($userPasswordHasher->hash  
    Password($user, $plainPassword));
```

L'entité User a été développée pour gérer toutes les informations du compte, tandis que les contrôleurs dédiés ont pris en charge les différentes actions utilisateur.

## Gestion des sujets

La fonctionnalité principale du site repose sur la possibilité de publier du contenu.

Pour cela, les éléments suivants ont été développés :

- Création d'un sujet
- Affichage d'un sujet en détail
- Liste des sujets triés par catégorie

## Système de commentaires

Chaque sujet permet aux utilisateurs d'interagir et d'échanger via un espace de commentaires.

### Fonctionnalités développées :

- Ajout de commentaires
- Modification par l'auteur ou un administrateur
- Suppression par un administrateur
- Affichage chronologique
- Lien direct entre User, Sujet et Comment

La structure a été pensée pour rendre les discussions fluides et lisibles.

*Exemple pour démontrer la relation entre User, Comment et Sujet dans l'entité Comment.*

```
1  /**
2   * Sujet auquel le commentaire est lié.
3   *
4   * @var Sujet|null
5   */
6  #[ORM\ManyToOne(targetEntity: Sujet::class, inversedBy: 'comments')]
7  #[ORM\JoinColumn(nullable: false)]
8  private ?Sujet $subject = null;
9
10 /**
11  * Utilisateur qui a écrit le commentaire (s'il est enregistré).
12  *
13  * @var User|null
14  */
15 #[ORM\ManyToOne(targetEntity: User::class, inversedBy: 'comments')]
16 #[ORM\JoinColumn(name: "user_id", referencedColumnName: "id", nullable: true, onDelete: "SET NULL")]
17 private ?User $authorUser = null;
```

## 4.3 Développement du frontend

### **Utilisation de Twig**

Le frontend a été développé avec Twig, le moteur de templates intégré à Symfony.

Les objectifs principaux :

- assurer une navigation intuitive,
- respecter une hiérarchie claire des pages,
- offrir une présentation adaptée à un site communautaire,
- garantir la cohérence visuelle.

### **Design et responsive**

Le site a été conçu pour être utilisable sur mobile, tablette et ordinateur :

- grille flexible,
- menus simplifiés,
- zones de texte adaptées,

---

## 4.4 Développement de l'administration

Une interface réservée aux administrateurs a été développée pour :

- gérer les utilisateurs (suppression, modification)
- gérer les sujets
- supprimer ou gérer les commentaires
- visualiser l'activité globale du site

Cette interface garantit le bon fonctionnement du site et le maintien d'un espace sain pour la communauté.

---

#### 4.5 Sécurité

Plusieurs mesures ont été mises en place :

- Protection des routes par rôles
- Validation des formulaires
- Filtrage des contenus utilisateur
- Tokens sécurisés pour les liens sensibles
- Vérification des permissions sur chaque action (ex. suppression d'un commentaire)

Ces mesures assurent un fonctionnement fiable et sécurisé, indispensable pour un site communautaire.

*Exemple du menu si l'utilisateur est connecté.*

```
1  {% if app.user %}  
2      <li>  
3          <a href="{{ path('forum') }}"  
4              class="{{ app.request.attributes.get('_route') ==  
'forum' ? 'active' : '' }}>  
5                  Forum  
6          </a>  
7      </li>  
8      <li>  
9          <a href="{{ path('profil') }}"  
10             class="{{ app.request.attributes.get('_route') ==  
'profil' ? 'active' : '' }}>  
11                 Mon profil  
12             </a>  
13         </li>  
14         <li>  
15             <a href="{{ path('app_logout') }}>Se déconnecter  
16         </a>  
17     {% else %}  
18         <li>  
19             <a href="{{ path('app_login') }}"  
20                 class="{{ app.request.attributes.get('_route') ==  
'app_login' ? 'active' : '' }}>  
21                 Se connecter  
22             </a>  
23         </li>  
24     {% endif %}
```

## 5. TESTS ET VALIDATION

### 5.1 Objectif des tests

La phase de tests a été essentielle pour garantir la fiabilité, la stabilité et la conformité du projet avec les besoins exprimés dans le cahier des charges.

L'objectif était de vérifier que :

- toutes les fonctionnalités développées fonctionnent correctement,
  - le site répond bien aux attentes fonctionnelles,
  - l'expérience utilisateur est fluide,
  - la sécurité est correctement assurée,
- 

### 5.2 Méthodologie utilisée

#### Tests manuels

Une grande partie de la validation fonctionnelle a été réalisée à travers des tests exploratoires et scénarios réels d'utilisation :

- création de comptes,
- connexion/déconnexion,
- création et modification de sujets,
- ajout/modification de commentaires,
- test des droits selon les rôles (utilisateur standard / administrateur),
- vérification des redirections,

Ces tests ont permis de corriger de nombreux détails d'ergonomie et de fonctionnement.

---

## Tests automatisés (PHPUnit)

Des tests unitaires ont été réalisés afin de valider le bon fonctionnement de certaines fonctionnalités critiques, notamment :

- la gestion des entités,
- la validation des formulaires,
- certains services internes.

Les tests automatisés permettent de garantir que les composants les plus sensibles du site restent stables malgré les évolutions du code.

*Extrait de test pour le formulaire de création de sujet.*

```
1  /**
2   * Test valide : vérifie si le formulaire accepte des don
3   * nées correctes.
4  */
5  public function testSubmitValidData(): void
6  {
7      // Données simulées valides
8      $formData = [
9          'username' => 'ValidUsername',
10         'email' => 'valid@example.com',
11         'plainPassword' => 'ValidPassword123',
12     ];
13
14     // Instancie une entité User vide pour la comparaison
15     $user = new User();
```

### 5.3 Validation de la sécurité

Plusieurs éléments ont été vérifiés :

- protection des routes par rôles
  - impossibilité d'accéder à des pages administrateur sans autorisation
  - vérification qu'un utilisateur ne peut pas modifier ou supprimer le contenu d'un autre utilisateur
  - filtrage des entrées utilisateur pour éviter les injections
  - fonctionnement correct de la réinitialisation de mot de passe
- 

### 5.4 Recette utilisateur

Une phase de recette a été réalisée en conditions réelles, simulant le comportement d'un utilisateur final :

- navigation complète sur le site,
- test de tout le parcours utilisateur,
- validation du bon affichage des pages et des messages d'erreur.

Cette recette a confirmé que le site répond aux attentes et fonctionne comme prévu.

---

## 5.5 Corrections et améliorations

Les tests ont permis d'identifier plusieurs points à corriger :

- amélioration de la mise en forme de certaines pages,
- correction de problèmes mineurs de redirection,
- ajustement des règles de validation des formulaires,
- renforcement de certaines vérifications de sécurité.

## 6. DOCUMENTATION

### 6.1 Objectif de la documentation

La documentation a pour but de fournir l'ensemble des informations nécessaires pour :

- comprendre le fonctionnement global du projet,
  - permettre à un développeur de maintenir ou faire évoluer l'application,
  - faciliter l'installation du projet dans un nouvel environnement,
  - détailler les fonctionnalités disponibles et leur usage,
  - offrir une vision claire de l'architecture et des choix techniques.
- 

### 6.2 Documentation technique

Architecture générale

La documentation décrit l'architecture du projet basée sur le framework Symfony :

- structure MVC
- organisation des contrôleurs
- organisation des entités et relations entre modèles
- rôle des services
- fonctionnement des templates Twig

Des schémas simplifiés ont été réalisés pour aider à visualiser la structure de l'application.

## Installation du projet

Un guide d'installation a été rédigé afin de permettre à un autre développeur d'installer rapidement le projet dans son environnement :

1. Cloner le projet via Git
2. Installer les dépendances avec Composer
3. Configurer l'environnement .env
4. Créer la base de données
5. Exécuter les migrations
6. Lancer le serveur Symfony
7. Configurer Mailtrap pour les emails

Ce guide permet une mise en place rapide et reproductible.

## *Extrait du fichier README*

```
1 ## Installation
2
3 Suivez ces étapes pour installer et lancer le projet localement :
4
5 ### 1. Clonez le dépôt
6
7 Cloner ce projet depuis GitHub :
8 ````bash
9 git clone https://github.com/marine1512/forum-app.git
10 cd forum-app
11 ````

12
13 ### 2. Installez les dépendances
14 Installez les dépendances PHP avec Composer :
15 ````bash
16 composer install
17 ````

18
19 Installez les dépendances front-end :
20 ````bash
21 npm install
22 npm run dev
23 ````

24
25 ### 3. Configurez le fichier ` `.env` :
26 ````bash
27 DATABASE_URL="mysql://admin:admin@127.0.0.1:3307/dev_app"
28 ````
```

## Description des entités

Pour chaque entité principale, la documentation détaille :

- sa structure
- ses champs
- ses relations
- son rôle dans le fonctionnement du site

## *Exemple de documentation de l'entité Catégorie.*

```
1  /**
2   * Entité représentant une catégorie.
3   *
4   * Une catégorie regroupe plusieurs sujets (entité `Sujet`)
5   * ) et peut être utilisée
6   * pour organiser les sujets dans des sections logiques.
7   */
8 #[ORM\Entity(repositoryClass: CategoryRepository::class)]
9 class Category
10 {
11     /**
12      * Identifiant unique de la catégorie.
13      *
14      * @var int|null
15      */
16     #[ORM\Id]
17     #[ORM\GeneratedValue]
18     #[ORM\Column(type: 'integer')]
19     private $id;
20
21     /**
22      * Nom de la catégorie.
23      *
24      * @var string|null
25      */
26     #[ORM\Column(type: 'string', length: 255)]
27     private $name;
```

### 6.3 Documentation utilisateur

Cette documentation explique, étape par étape, comment utiliser le *Forum Fan de Lore*.

Elle s'adresse aux utilisateurs qui souhaitent consulter des discussions, créer un compte, participer aux échanges ou interagir avec la communauté.

#### **Accéder au site**

1. Ouvrir un navigateur web (Chrome, Firefox, Edge...).
2. Entrer l'adresse du site dans la barre d'URL.
3. Arriver sur la page d'accueil présentant les dernières discussions.

#### **Créer un compte**

Certains éléments sont visibles sans compte, mais la participation nécessite une inscription.

#### **Étapes :**

1. Cliquer sur le bouton « S'inscrire » en haut du site.
2. Remplir :
  - un pseudo unique,
  - une adresse e-mail valide,
  - un mot de passe.
3. Valider le formulaire.
4. Confirmer l'adresse e-mail via un lien reçu.
5. Le compte est créé.

*Exemple de création de formulaire pour l'inscription.*

```
1  public function buildForm(FormBuilderInterface $builder,
2      array $options): void
3  {
4      $builder
5          ->add('username')
6          ->add('email')
7          ->add('plainPassword', PasswordType::class, [
8              'mapped' => false,
9              'attr' => ['autocomplete' => 'new-password'],
10             'constraints' => [
11                 new NotBlank([
12                     'message' => 'Please enter a password',
13                 ]),
14                 new Length([
15                     'min' => 6,
16                     'minMessage' => 'Your password should
be at least {{ limit }} characters',
17                     'max' => 4096,
18                     ]),
19             ]);
20 }
```

## Se connecter

1. Cliquer sur « Connexion ».
2. Saisir email et mot de passe.
3. Cliquer sur « Se connecter ».
4. L'utilisateur est redirigé vers l'accueil du forum (selon son rôle)

## Naviguer dans le forum

1. Choisir une catégorie dans le menu ou la page d'accueil.
2. Consulter la liste des sujets.
3. Cliquer sur un sujet pour lire les messages.

## Créer un sujet

Pour ouvrir une discussion dans une catégorie :

1. Cliquer sur « Nouveau sujet ».
2. Saisir :
  - un titre clair,
  - choisir une catégorie
3. Cliquer sur « Créer le sujet ».
4. Le sujet apparaît immédiatement dans la liste selon la catégorie choisie.

## Rédiger un commentaire

1. Ouvrir un sujet.
2. Descendre jusqu'à la zone dédiée.
3. Écrire son message.
4. Cliquer sur « Publier ».
5. La réponse apparaît dans la discussion.

## Modifier ses messages (si vous en êtes l'auteur)

L'utilisateur peut corriger ou retirer ses propres publications.

1. Cliquer sur le bouton "modifier" au-dessus du message.
2. Pour la modification : mettre à jour le texte puis cliquer sur « Enregistrer ».

## Voir son profil

1. Cliquer sur son pseudo au niveau de la page d'accueil ou accéder à la page « Mon profil ».

## Déconnexion

1. Cliquer sur le bouton « Déconnexion » dans le menu.
  2. L'utilisateur est redirigé vers la page d'accueil.
- 

## 6.4 Documentation Pour l'administrateur

### Accéder au panneau d'administration

1. Se connecter en tant qu'administrateur.
2. Cliquer sur le lien « Tableau administrateur » présent dans le menu.
3. Le tableau de bord donne une vue générale du forum :
  - nombre d'utilisateurs,
  - nombre de catégories,
  - nombre de sujets,
  - nombre de commentaires.

*Exemple de la card user présente sur le tableau administrateur.*

```
1 <div class="row g-3 justify-content-center admin-cards">
2   <div class="col-12 col-md-6 col-lg-3">
3     <div class="card text-center h-100">
4       <div class="card-body d-flex flex-column justify-content-between">
5         <div>
6           <div class="h2">{{ stats.users }}</div>
7           <div class="text-muted mb-3">Membres</div>
8         </div>
9         <a class="bouton" href="{{ path('admin_members') }}>Voir</a>
10        </div>
11      </div>
12    </div>
```

## Gérer les utilisateurs

### Consulter la liste des utilisateurs

1. Cliquer sur « membres » dans le menu d'administration.
2. La liste complète apparaît avec :
  - pseudo
  - email
  - mot de passe
  - boutons Modifier et Supprimer

### Modifier un utilisateur

1. Cliquer sur « Modifier » à côté d'un utilisateur.
2. Changer : pseudo, email ou mot de passe.
3. Cliquer sur « Enregistrer ».

## Supprimer un utilisateur

1. Cliquer sur « Supprimer »
  2. Confirmer l'action.
- 

## Gérer les catégories du forum

### Liste des catégories

1. Cliquer sur « catégories ».
2. La liste affiche :
  - nom
  - boutons Modifier et Supprimer

### Ajouter une catégorie

1. Aller dans « Catégories ».
2. Cliquer sur « Nouvelle catégorie ».
3. Renseigner :
  - nom,
4. Cliquer sur « Créer ».

### Modifier une catégorie

1. Dans la liste des catégories, cliquer sur « Modifier ».
2. Changer les informations.
3. Cliquer sur « Mettre à jour ».

### Supprimer une catégorie

1. Cliquer sur « Supprimer ».
  2. Confirmer.
-

### *Exemple de suppression d'une catégorie.*

```
1  #[Route('/categories/{id}/delete', name: 'categories_delete',
2  methods: ['POST'])]
3  public function categoriesDelete(Category $category, Request
4  $request, EntityManagerInterface $em): Response
5  {
6      if ($this->isCsrfTokenValid('delete_category_'. $category-
7      >getId(), $request->request->get('_token'))) {
8          $em->remove($category);
9          $em->flush();
10         $this->addFlash('success', 'Catégorie supprimée.');
11     } else {
12         $this->addFlash('error', 'Token CSRF invalide.');
13     }
14 }
```

## Gérer les sujets

### Liste des sujets

1. Cliquer sur « Sujets ».
2. La liste affiche :
  - o nom
  - o la catégorie à laquelle elle appartient.
  - o boutons Modifier et Supprimer

## **Modifier un sujet**

1. Cliquer sur « Modifier ».
2. Changer le titre ou le contenu si nécessaire.
3. Enregistrer.

## **Supprimer un sujet**

1. Cliquer sur « Supprimer ».
  2. Confirmer.
- 

## **Modération des commentaires**

### **Liste des commentaires**

1. Cliquer sur « commentaires ».
2. La liste affiche :
  - auteur
  - extrait
  - boutons Modifier et Supprimer

### **Modifier un commentaire**

1. Cliquer sur « Modifier ».
2. Changer le titre ou le contenu si nécessaire.
3. Enregistrer.

## **Supprimer un commentaire**

1. Cliquer sur « Supprimer ».
2. Confirmer.

## *Exemple de création de formulaire pour la modification d'un commentaire.*

```
1  public function buildForm(FormBuilderInterface $builder, array $options): void
2  {
3      $builder
4          ->add('text', TextareaType::class, [
5              'label' => 'Modifier votre commentaire',
6              'constraints' => [
7                  new \Symfony\Component\Validator\Constraints\NotBlank([
8                      'message' => 'Le commentaire ne peut pas être vide.',
9                  ]),
10                 new \Symfony\Component\Validator\Constraints\Length([
11                     'min' => 5,
12                     'minMessage' => 'Le commentaire doit comporter au moins
{{ limit }} caractères.',
13                     'max' => 2000,
14                     'maxMessage' => 'Le commentaire ne peut pas dépasser {{
limit }} caractères.',
15                     ]),
16                 ],
17             ]);
18 }
```

## 7. BILAN PERSONNEL

### 7.1 Compétences acquises

Ce projet a représenté une étape importante dans mon parcours de formation. Il m'a permis d'appliquer concrètement les connaissances théoriques apprises, et d'acquérir de nouvelles compétences, notamment :

- Maîtrise du framework Symfony et de son architecture MVC
- Conception d'une base de données relationnelle cohérente
- Développement d'un système d'authentification complet (sécurité, rôles, permissions)
- Mise en place d'un espace utilisateur et d'un module de publication
- Création d'une interface d'administration permettant la gestion et la modération
- Gestion d'un projet web de bout en bout, du cahier des charges aux tests
- Clarté et rigueur dans la documentation et la structure du code

Ce projet a renforcé ma confiance en mes capacités à mener un développement complet en autonomie.

---

## 7.2 Difficultés rencontrées

Plusieurs difficultés ont été rencontrées durant le développement :

- la gestion des relations entre les entités au sein de Doctrine,
- la mise en place des règles de sécurité selon les rôles,
- l'organisation du code pour garder une application claire et modulaire,
- la gestion des erreurs et validations de formulaires,
- l'intégration cohérente du front (Twig) avec la logique backend.

Ces obstacles ont nécessité des recherches, des tests et parfois une réorganisation du code, ce qui a été très formateur.

---

## 7.3 Améliorations possibles

Même si le projet remplit ses objectifs, plusieurs pistes d'amélioration peuvent être envisagées :

- ajouter un système de notifications utilisateur,
- intégrer un système de "likes" ou de mise en favoris,
- mettre en place une modification du profil utilisateur,
- optimiser le design et la performance globale du site,
- mettre en place des tests unitaires plus nombreux et des tests fonctionnels.

Ces évolutions seraient pertinentes pour une version plus aboutie du projet.

---

## 8. CONCLUSION

Ce projet a été une expérience extrêmement enrichissante, tant sur le plan technique que personnel.

Il m'a permis de mettre en pratique l'ensemble des compétences acquises durant ma formation, mais aussi d'en développer de nouvelles, notamment en matière d'architecture logicielle, d'autonomie et de gestion de projet.

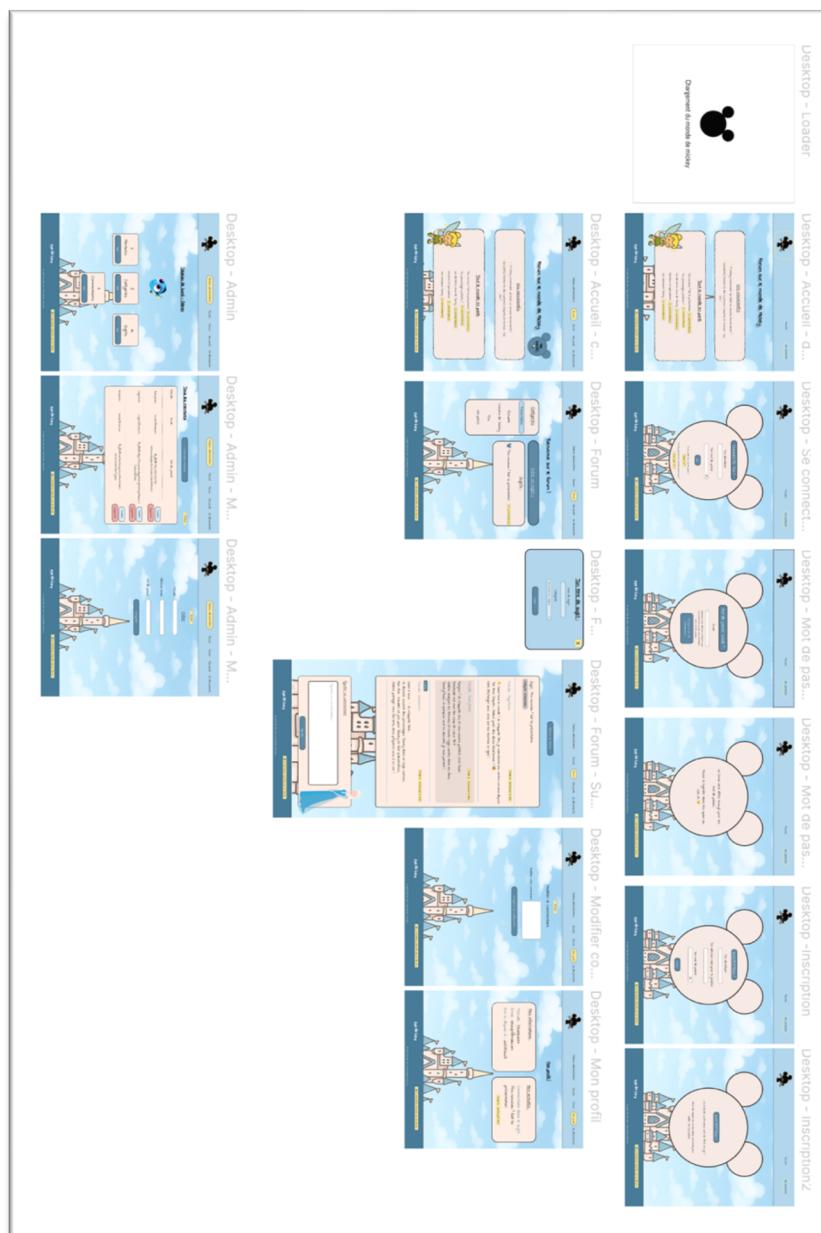
Le site web réalisé me semble répondre pleinement aux objectifs fixés : il propose un espace dédié aux fans du monde de Mickey, permet l'interaction entre utilisateurs et offre une interface d'administration complète pour assurer la modération et la gestion des contenus.

Au-delà de l'aspect technique, ce projet m'a permis de confirmer mon intérêt pour le développement web et ma volonté de continuer à progresser dans ce domaine.

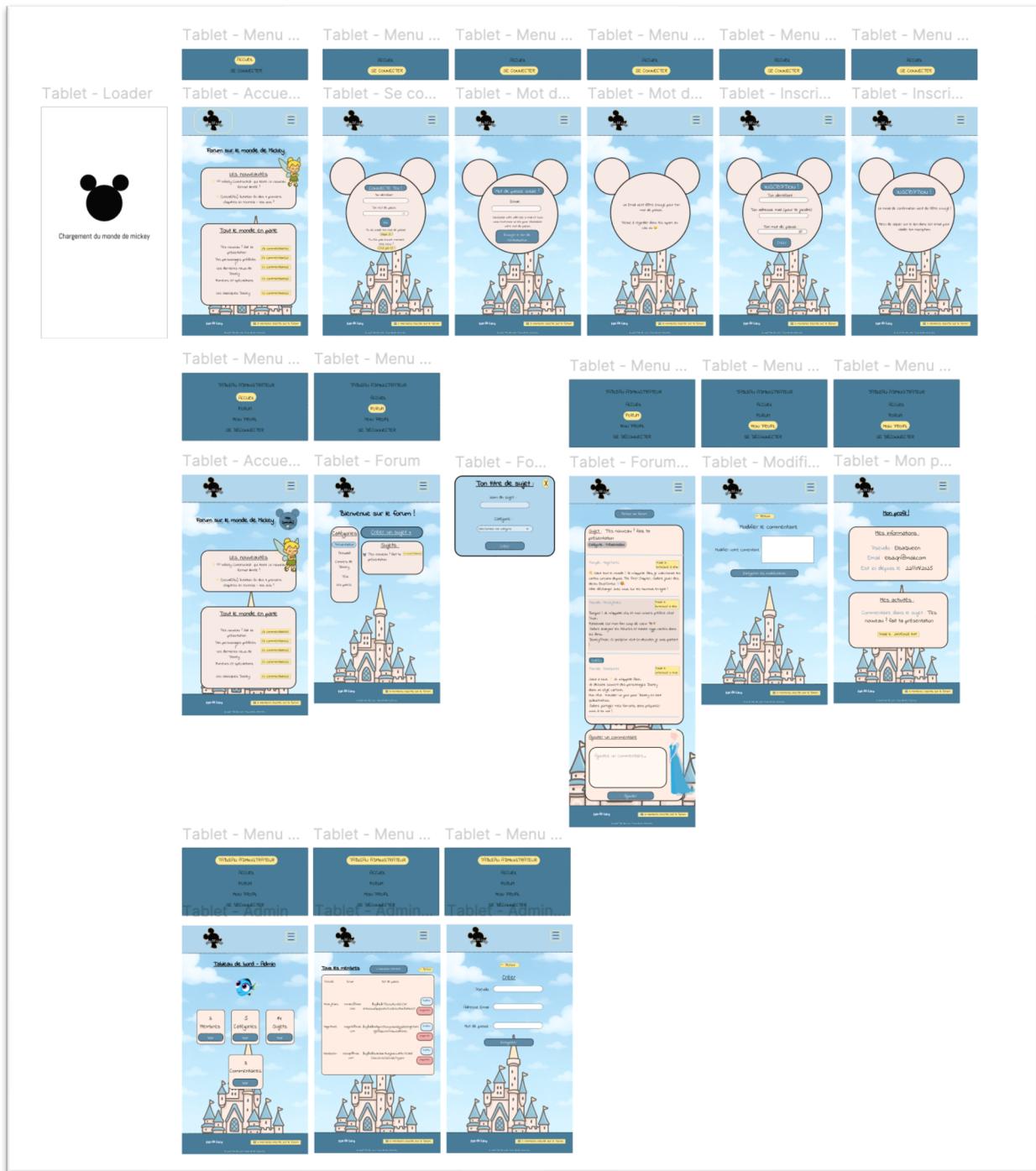
Il représente une étape importante dans l'obtention de mon Titre Professionnel et constitue une base solide pour mes futurs projets professionnels.

## 9. ANNEXES

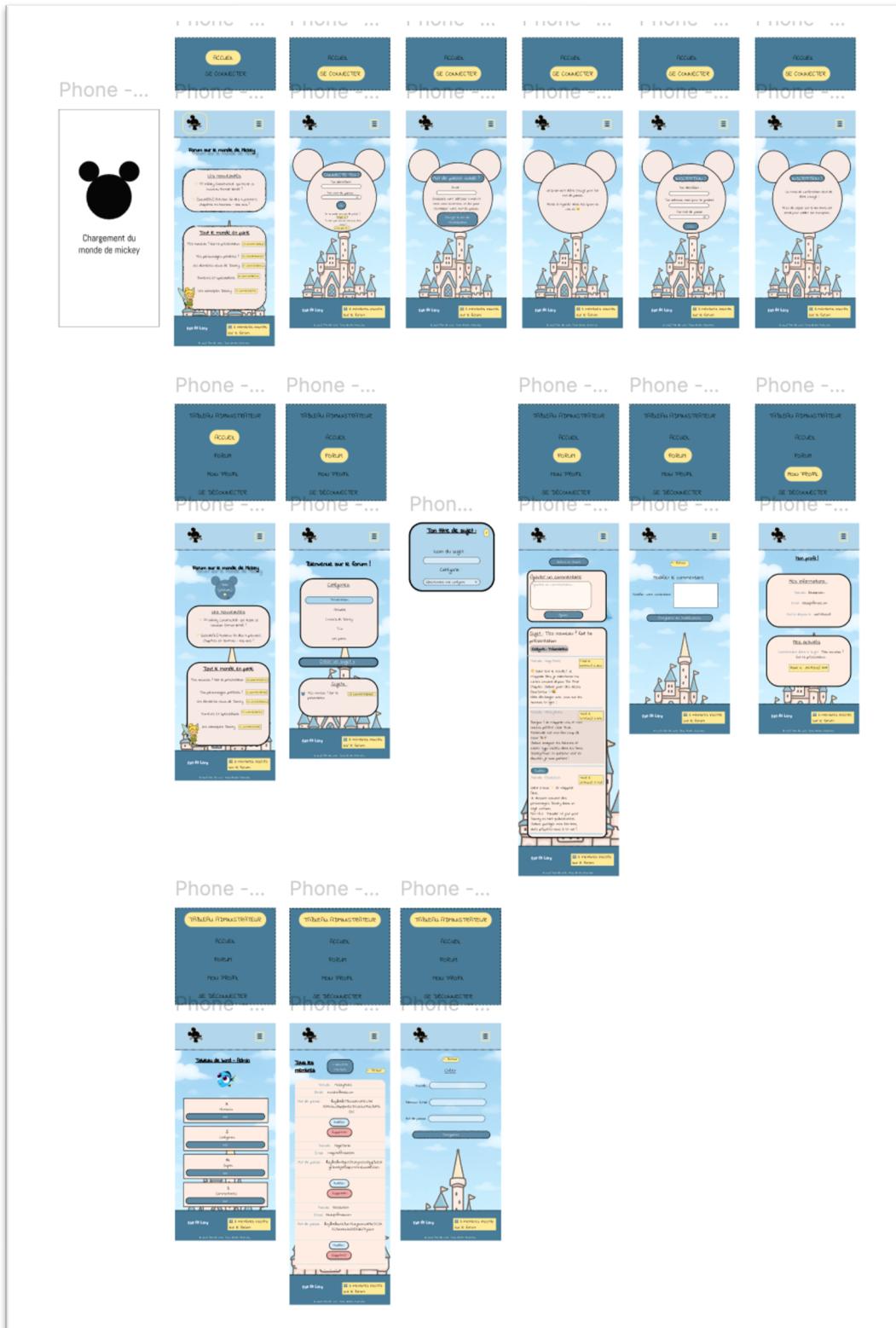
### *Maquettes Ordinateur*



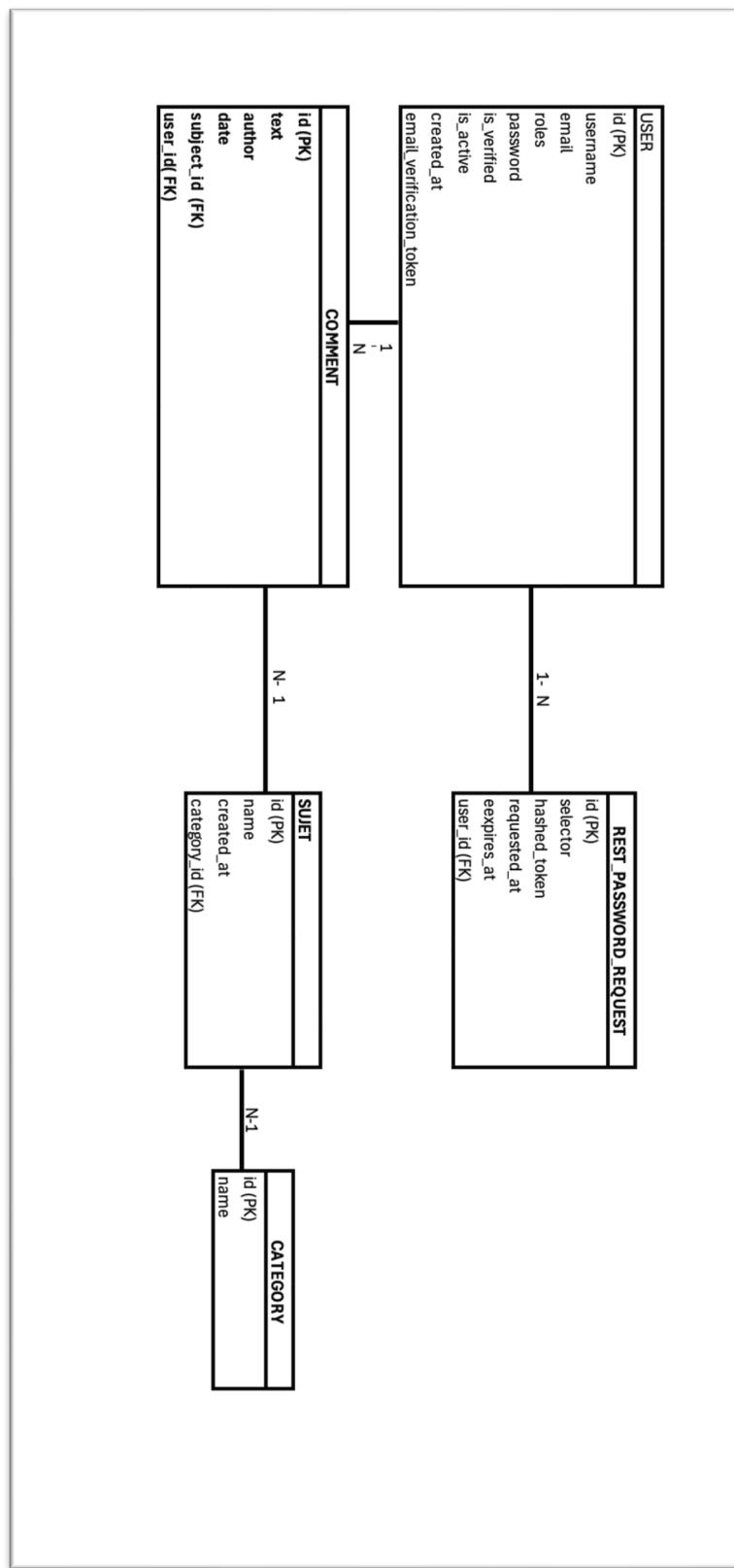
## Maquettes Tablette



## Maquettes Mobile



*Graphique MLD*



## Create-user part 1

```
1 #[AsCommand(
2     name: 'app:create-user',
3     description: 'Créer un membre (USER) avec son nom, email et mot de passe',
4     e.']
5 ])
6 class CreateUserCommand extends Command
7 {
8     private EntityManagerInterface $entityManager;
9     private UserPasswordHasherInterface $passwordHasher;
10
11    public function __construct(EntityManagerInterface $entityManager, User
12     asswordHasherInterface $passwordHasher)
13    {
14        parent::__construct();
15
16        $this->entityManager = $entityManager;
17        $this->passwordHasher = $passwordHasher;
18    }
19
20    protected function configure(): void
21    {
22        $this
23            ->addOption('username', null, InputOption::VALUE_REQUIRED, 'Nom
24             d\'utilisateur de l\'administrateur')
25            ->addOption('email', null, InputOption::VALUE_REQUIRED, 'Email
26             e l\'administrateur')
27            ->addOption('password', null, InputOption::VALUE_REQUIRED, 'Mot
28             de passe de l\'administrateur');
29    }
30
31    protected function execute(InputInterface $input, OutputInterface $outp
32     ut): int
33    {
34        $io = new SymfonyStyle($input, $output);
35
36        $username = $input->getOption('username');
37        $email = $input->getOption('email');
38        $password = $input->getOption('password');
39
40        $user = $this->entityManager->getRepository(User::class)->find($username);
41
42        if ($user) {
43            $this->output->writeln("L'utilisateur existe déjà.");
44            return 1;
45        }
46
47        $user = new User();
48        $user->setUsername($username);
49        $user->setEmail($email);
50        $user->setPassword($this->passwordHasher->hash($password));
51
52        $this->entityManager->persist($user);
53        $this->entityManager->flush();
54
55        $this->output->writeln("L'utilisateur a été créé avec succès.");
56
57        return 0;
58    }
59}
```

## Create-user part 2

```
1      if (!$username || !$email || !$password) {
2          $io->error('Vous devez fournir un nom d\'utilisateur (--username), un ema
3              il (--email) et un mot de passe (--password).');
4          return Command::FAILURE;
5
6      $existingUser = $this->entityManager->getRepository(User::class)->findOneBy
7 ([['email' => $email]]);
8      if ($existingUser) {
9          $io->error('Un utilisateur avec cet email existe déjà.');
10         return Command::FAILURE;
11     }
12
13     $user = new User();
14     $user->setUsername($username)
15         ->setEmail($email)
16         ->setRoles(['ROLE_USER'])
17         ->setPassword($this->passwordHasher->hashPassword($user, $password))
18         ->setIsActive(true)
19         ->setIsVerified(true);
20
21     $this->entityManager->persist($user);
22     $this->entityManager->flush();
23
24     $io->success('Le membre a été créé avec succès !');
25     $io->writeln(sprintf('Nom d\'utilisateur: %s', $username));
26     $io->writeln(sprintf('Email: %s', $email));
27
28     return Command::SUCCESS;
29 }
```

## *Controller d'inscription part 1 (avec documentation)*

```
1  /**
2   * Contrôleur pour gérer l'inscription des utilisateurs.
3   */
4  class RegisterController extends AbstractController
5  {
6      private EmailVerifier $emailVerifier;
7
8      public function __construct(EmailVerifier $emailVerifier)
9      {
10         $this->emailVerifier = $emailVerifier;
11     }
12
13     /**
14      * Gère l'inscription des nouveaux utilisateurs.
15      *
16      * @param Request $request La requête HTTP.
17      * @param UserPasswordHasherInterface $userPasswordHasher Le service de hachage d
e mot de passe.
18      * @param EntityManagerInterface $entityManager Le gestionnaire d'entités pour la
persistance des données.
19      * @return Response La réponse HTTP contenant la vue d'inscription ou une redirec
tion.
20      */
21     #[Route('/register', name: 'app_register')]
22     public function register(Request $request, UserPasswordHasherInterface $userPassw
ordHasher, EntityManagerInterface $entityManager): Response
23     {
24         $user = new User();
25         $form = $this->createForm(RegistrationForm::class, $user);
26         $form->handleRequest($request);
27
28         if ($form->isSubmitted() && $form->isValid()) {
29             $plainPassword = $form->get('plainPassword')->getData();
30
31             $user->setPassword($userPasswordHasher->hashPassword($user, $plainPasswor
d));
32
33             $user->setRoles(['ROLE_USER']);
34     }
```

## *Controller d'inscription part 2 (avec documentation)*

```
1      $emailVerificationToken = Uuid::v4()->toRfc4122();
2      $user->setEmailVerificationToken($emailVerificationToken);
3
4      $entityManager->persist($user);
5      $entityManager->flush();
6
7      $emailVerificationUrl = $this->generateUrl(
8          'app_verify_email', // Nom de la route
9          ['token' => $emailVerificationToken], // Paramètres de la route
10         UrlGeneratorInterface::ABSOLUTE_URL // URL absolue
11     );
12
13     $htmlEmailVerificationUrl = htmlspecialchars($emailVerificationUrl);
14
15     $this->emailVerifier->sendEmailConfirmation('app_verify_email', $user,
16         (new TemplatedEmail())
17             ->from(new Address('mignomarine@gmail.com', 'Mail Bot'))
18             ->to((string) $user->getEmail())
19             ->subject('Merci de confirmer votre mail')
20             ->html(
21                 '<h1>Bienvenue sur notre application !</h1>' .
22                 '<p>Merci de vous être inscrit. Veuillez confirmer votre adre
sse email en cliquant sur le lien suivant :</p>' .
23                 '<a href="' . $htmlEmailVerificationUrl . '">Confirmer mon ad
resse</a>' )
24             )
25     );
```

### *Controller d'inscription part 3 (avec documentation)*

```
1      $this->addFlash('success', 'Un email de confirmation a été envoyé. Veuillez vérifier votre boîte de réception.');
2
3      return $this->redirectToRoute('app_register_confirmation');
4  }
5
6  return $this->render('security/inscription.html.twig', [
7      'registrationForm' => $form,
8  ]);
9 }
10
11 /**
12 * Vérifie l'adresse email de l'utilisateur à l'aide du token fourni.
13 *
14 *
15 * @param Request $request La requête HTTP.
16 * @param EntityManagerInterface $entityManager Le gestionnaire d'entités pour la persistance des données.
17 * @param UserAuthenticatorInterface $userAuthenticator Le service d'authentification des utilisateurs.
18 * @param FormLoginAuthenticator $authenticator L'authentificateur de connexion par formulaire.
19 * @return Response La réponse HTTP après la vérification.
20 */
21 #[Route('/verify-email', name: 'app_verify_email')]
22 public function verifyUserEmail(Request $request, EntityManagerInterface $entityManager, UserAuthenticatorInterface $userAuthenticator, FormLoginAuthenticator $authenticator): Response
23 {
24     $token = $request->query->get('token');
25
26     if (!$token) {
27         throw $this->createNotFoundException('Token non fourni pour la vérification.');
28     }
}
```

### *Controller d'inscription part 4 (avec documentation)*

```
1      $user = $entityManager->getRepository(User::class)->findOneBy(['emailVerificationToken' => $token]);
2
3      if (!$user) {
4          throw $this->createNotFoundException('Aucun utilisateur trouvé pour ce token.');
5      }
6      if (!$user->isVerified()) {
7          $user->setIsVerified(true);
8          $user->setIsActive(true);
9          $user->setEmailVerificationToken(null);
10         $entityManager->flush();
11     }
12     try {
13         return $userAuthenticator->authenticateUser(
14             $user,
15             $authenticator,
16             $request
17         );
18     } catch (AuthenticationException $e) {
19         $this->addFlash('error', 'Une erreur est survenue lors de l\'authentification automatique.');
20         return $this->redirectToRoute('app_login');
21     }
22
23     $this->addFlash('success', 'Votre email a été confirmé avec succès et vous êtes maintenant connecté.');
24     return $this->redirectToRoute('home');
25 }
26
27 /**
28  * Affiche une page de confirmation après l'inscription.
29  *
30  * @Route("/register/confirmation", name="app_register_confirmation")
31  * @return Response La réponse HTTP contenant la vue de confirmation.
32  */
33 #[Route('/register/confirmation', name: 'app_register_confirmation')]
34
35 public function confirmEmailNotification(): Response
36 {
37     return $this->render('security/confirmation_email.html.twig');
38 }
39 }
```